Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«Уральский федеральный университет имени первого Президента России Б.Н.Ельцина»
Институт радиоэлектроники и информационных технологий – РТФ

Анализ сложности алгоритмов сортировки строк

Отчёт по лабораторной работе

по дисциплине «Алгоритмы, структуры данных и анализ сложности»

Вариант 1

Выполнил: Тарасенко А. Р.
студент гр. РИ-230913
Преподаватель:

Доцент, к.ф. -м.н.
Трофимов С.П.

Екатеринбург, 2024

Оглавление

Задание

Реализовать один из алгоритмов сортировки строк:

1. Пузырьковая сортировка BubbleSort.

Выбор алгоритма выбирается по согласованию с преподавателем.

Для алгоритма определить сложность относительно наиболее характерной операции (сравнение, перестановка и др.). Вид функции сложности F(n) подобрать в соответствии с теорией. Например, для оптимальных алгоритмов $F(n) = C*n*\log_2(n)$. Найти также коэффициент пропорциональности C. Для аппроксимации можно использовать метод наименьших квадратов и сервис «Поиск решения».

План проведения эксперимента с алгоритмом называется массовой задачей. Представьте план в виде xml-файла.

Результаты решения массовой задачи записать в текстовый файл в 2 столбика: длина массива, количество операций. Файл импортировать в Excel. В «шапке» листа указать параметры тренда, вычислить квадратичные невязки и минимизировать их сумму с помощью «Данные-Поиск решения»

Теоретическая часть

Алгоритм: Сортировка пузырьком

Пузырьковая сортировка работает так:

многократно проходим по массиву,

на каждой итерации попарно сравниваем соседние элементы,

если элементы стоят "неправильно" — меняем их местами.

Повторяем процесс до тех пор, пока массив не отсортирован.

Алгоритм сортировки будет состоять из двух основных операций:

1) if (array[i] > array[i + 1]) сравнение элементов массива.
2) (array[i], array[i + 1]) = (array[i + 1], array[i]) – смена элементов массива местами.

Особенности алгоритма:

1) По сравнению с другими алгоритмами считается простейшим для понимания и реализации.
2) Эффективен для массивов небольшого размера.

Сложность Алгоритма:

- На первой итерации нужно сделать n−1n-1n−1 сравнений.
- На второй итерации — n−2n-2n−2 сравнений.
- И так далее, вплоть до одного сравнения.

Итого количество сравнений:

(n-1)+(n-2)…+2+1 = n(n-1)/2

(Используемый мной алгоритм оптимизирован так, что если после одной из проверок не было произведено перестановок, то алгоритм завершается. Тем самым в лучшем случае количество сравнений = n-1.

Количество перестановок:

В худшем случае, т.е. когда массив отсортирован в обратном порядке - столько же, сколько и сравнений. В лучшем случае – 0. Их сумма примерно равна n^2.

Инструкция пользователя

При запуске программы открывается консоль, где пользователю предлагается перед сортировкой собственного массива провести эксперимент и оценить производительность алгоритма. Если он согласен, то на консоль будут выведены результаты эксперимента.

```
Выполнить эксперимент перед началом программы (yes/no)? yes
Arithmetic progression:
Длина массива: 1000     Кол-во массивов: 15     Операторов 'if' в среднем: 498198       Свапов в среднем:248098
Длина массива: 1100     Кол-во массивов: 15     Операторов 'if' в среднем: 498679       Свапов в среднем:249833
Длина массива: 1200     Кол-во массивов: 15     Операторов 'if' в среднем: 498746       Свапов в среднем:247532
Длина массива: 1300     Кол-во массивов: 15     Операторов 'if' в среднем: 498154       Свапов в среднем:250076
Длина массива: 1400     Кол-во массивов: 15     Операторов 'if' в среднем: 498498       Свапов в среднем:249017
Длина массива: 1500     Кол-во массивов: 15     Операторов 'if' в среднем: 498579       Свапов в среднем:245140
Длина массива: 1600     Кол-во массивов: 15     Операторов 'if' в среднем: 498485       Свапов в среднем:248648
Длина массива: 1700     Кол-во массивов: 15     Операторов 'if' в среднем: 498568       Свапов в среднем:248067
Длина массива: 1800     Кол-во массивов: 15     Операторов 'if' в среднем: 498568       Свапов в среднем:248370
Длина массива: 1900     Кол-во массивов: 15     Операторов 'if' в среднем: 498317       Свапов в среднем:250601
Длина массива: 2000     Кол-во массивов: 15     Операторов 'if' в среднем: 498055       Свапов в среднем:248879
Длина массива: 2100     Кол-во массивов: 15     Операторов 'if' в среднем: 498369       Свапов в среднем:249073
Длина массива: 2200     Кол-во массивов: 15     Операторов 'if' в среднем: 498518       Свапов в среднем:248984
Длина массива: 2300     Кол-во массивов: 15     Операторов 'if' в среднем: 498648       Свапов в среднем:250741
Длина массива: 2400     Кол-во массивов: 15     Операторов 'if' в среднем: 498310       Свапов в среднем:249766
Длина массива: 2500     Кол-во массивов: 15     Операторов 'if' в среднем: 498524       Свапов в среднем:249091
Длина массива: 2600     Кол-во массивов: 15     Операторов 'if' в среднем: 498611       Свапов в среднем:248409
Длина массива: 2700     Кол-во массивов: 15     Операторов 'if' в среднем: 498840       Свапов в среднем:251205
Длина массива: 2800     Кол-во массивов: 15     Операторов 'if' в среднем: 498522       Свапов в среднем:249509
Длина массива: 2900     Кол-во массивов: 15     Операторов 'if' в среднем: 498492       Свапов в среднем:249112
Длина массива: 3000     Кол-во массивов: 15     Операторов 'if' в среднем: 498327       Свапов в среднем:248528
```

Рисунок 1 – Проведение эксперимента.

Далее, независимо от того, производился опыт или нет, пользователю предлагается ввести собственный массив строк, разделяя его элементы пробелом. После ввода на консоль будет выведен уже отсортированный массив.

```
Введите строки для сортировки, разделяя их пробелом или 'Enter' для завершения: 505 1984 1255 14 52 42 39 69
Отсортированный массив: 14 39 42 52 69 505 1255 1984

Введите строки для сортировки, разделяя их пробелом или 'Enter' для завершения:
PS D:\Learn\Алгосы\BubbleSort>
```

Рисунок 2 – Сортировка пользовательского массива чисел.

Инструкция программиста

В программе, написанной на языке C#, алгоритм пирамидальной сортировки представлен в классе BubbleSort. Для его реализации была написана функция Sort:

public static int[] Sort(int[] array)

принимающая массив строк и сортирующая его алгоритмом Bubble Sort. Для возможности оценивать производительность сортировки функция возвращает значение счётчика обоих операций – сравнивания и перестановки, функция изменяет сам массив из-за чего она не возвращает его.

Также в программе реализована функция GetOperationsCount из класса ArrayGenerator:

public int[] GetOperationsCount(int length, int minElement, int maxElement, int repeat)

принимающая параметры для проведения эксперимента и проводя его.

Функция static void Experiments() класса Program которая парсит параметры из файла xml и запускает вышеуказанную функцию GetOperationsCount для получения результатов

Тестирование

Кроме основных и вспомогательных, программа также содержит функцию Test. Она содержит тесты, проверяющий корректность работы алгоритма сортировки и выполняется в момент запуска программы. Если один или несколько тестов завершились неудачно, на консоль будут выведены соответствующие сообщения, и программа завершит работу. Часть кода тестов представлена ниже, с полным кодом можно ознакомиться в Приложении.

```
1 reference
static bool testing(){

    //Test 1

    bool is_passed = true;
    var array = new int[] { 5, 4, 3, 2, 1 };
    var true_array = new int[] {1, 2, 3, 4, 5};
    BubbleSort.Sort(array);

    if (!array.SequenceEqual(true_array)){
        Console.WriteLine($"Test 1 isn't passed.");
        Console.WriteLine("Actual:");
        foreach (var item in array) Console.Write(item + " ");
        Console.Write("\n\n");

        Console.WriteLine("Expected:");
        foreach (var item in true_array) Console.Write(item + " ");
        Console.Write("\n\n");

        is_passed = false;
    }

    //Test 2
    array = new int[]{1, 2, 3, 4, 5};
    true_array = new int[] {1, 2, 3, 4, 5};
    BubbleSort.Sort(array);

    if (!array.SequenceEqual(true_array)){
        Console.WriteLine($"Test 2 isn't passed.");
        Console.WriteLine("Actual:");
        foreach (var item in array) Console.Write(item + " ");
        Console.Write("\n\n");

        Console.WriteLine("Expected:");
        foreach (var item in true_array) Console.Write(item + " ");
        Console.Write("\n\n");

        is_passed = false;
    }
```

Рисунок 3 - Часть кода тестирующей функции Test

Помимо тестирования, программа предоставляет возможность провести эксперимент с пузырьковой сортировкой на больших массивах данных. Для этого в xml-файле main был создан план эксперимента. Он включает в себя несколько элементов nodes, каждый из которых описывает свою часть эксперимента: какое количество массивов будет сгенерировано, какой длины, с какими значениями и по какому принципу эти массивы будут изменяться (здесь реализовано изменения длины массива в арифметической и геометрической прогрессиях).

Данный документ обрабатывается в функции Experiments: значения атрибутов каждого элемента эксперимента будут получены и использованы для генерации массивов (все значения их элементов выбираются случайным образом из заданного диапазона) и последующей сортировки.

Полученные результаты (длины массивов и кол-во операций для их сортировки) заносятся в таблицу Excel. В добавление к ним рассчитываем также значения тренда для каждого случая и строим графики функции сложности. Скорее всего, они будут отстоять далеко друг от друга, поэтому для их сближения применим метод наименьших квадратов: рассчитываем для каждого случая квадратичные невязки, а затем получим коэффициент для функции сложности с помощью сервиса Excel «Поиск решения».

График функции сложности

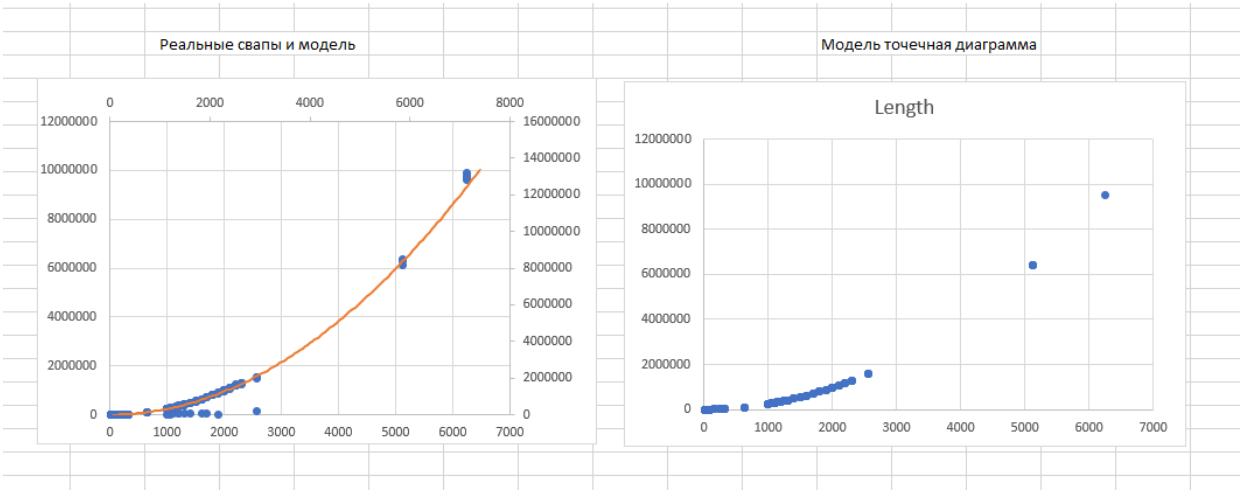

Рисунок 4 - Графики функции сложности после применения сервиса «Поиск решения»

В нашем случае для пузырьковой сортировки мы получаем коэффициент пропорциональности C = 0,246384028967765, а функция сложности для данного алгоритма принимает вид F(n) = 0,25 * n^2.

Выводы

В данной работе мы познакомились с одним из алгоритмов сортировки - пирамидальной сортировкой BubbleSort, написали программу для сортировки массивов чисел с его использованием, а также провели эксперимент для оценки сложности данного алгоритма. Полученная программа работает исправно и позволяет достаточно быстро сортировать массивы из тысяч строк. Это же подтверждается и результатами эксперимента: алгоритм имеет сложность вида O (n^2).

Сортировка пузырьком не является эффективной, однако является простой для изучения в рамках программы изучения университетской программы.

Литература

1. Книги:

1) Левитин А. В. *Алгоритмы. Введение в разработку и анализ*. — М.: Вильямс, 2006. — 576 с. — Гл. 3. Метод грубой силы: Пузырьковая сортировка. — С. 144–146.
2) Федоряева Т. И. *Комбинаторные алгоритмы: учебное пособие*. — Новосибирск: Новосибирский гос. ун-т, 2011. — 118 с.

2. Электронные ресурсы:

3) Сортировка пузырьком // Википедия : [сайт]. — URL: https://ru.wikipedia.org/wiki/Сортировка_пузырьком (дата обращения: 08.05.2025).
4) Пузырьковая сортировка // Habr : [сайт]. — URL: https://habr.com/ru/articles/204600/ (дата обращения: 08.05.2025).

Приложение А

Ссылка на исходный код

Ссылка на репозиторий с реализацией алгоритма пузырьковой сортировки:
GitHub: https://github.com/delilit/BubbleSort

Приложение Б

Программный код

```
using System;
using System.Xml;
using System.Xml.Linq;
//using Лаб2_Сортировка_строк;

namespace SortingAlgorithms
{
   public class BubbleSort
   {
      public static int[] Sort(int[] array)
      {
         var IfCount = 0;
         var SwapsCount = 0;
         for (int j = 1; j < array.Length; j++)
         {
            bool isSorted = true;
            for (int i = 0; i < array.Length - j; i++)
            {
               IfCount++;
               if (array[i] > array[i + 1])
               {
                  SwapsCount++;
                  (array[i], array[i + 1]) = (array[i + 1], array[i]);
                  isSorted = false;
               }
            }
```

```
                if (isSorted)
                    break;
            }
            return [IfCount, SwapsCount];
        }
    }

    public class ArrayGenerator
    {
        private readonly Random _random = new Random();

        public int[] GetOperationsCount(int length, int minElement, int
maxElement)
        {
            int[] result = new int[2];
            int[] array = new int[length];
                for (int i = 0; i < length; i++)
                {
                    array[i] = _random.Next(minElement, maxElement + 1);
                }
            int[] sortResult = BubbleSort.Sort(array);
            return (sortResult);
        }

    public class Program
    {
        static void Experiments()
        {
            try
            {
```

```
                var xmlDoc = XDocument.Load("main.xml");
                var generator = new ArrayGenerator();

                foreach (var node in xmlDoc.Descendants("nodes"))
                {
                    string name = node.Attribute("name")?.Value ?? string.Empty;
                    int startLength = int.Parse(node.Attribute("startLength")?.Value
?? "0");
                    int maxLength = int.Parse(node.Attribute("maxLength")?.Value
?? "0");
                    int minElement =
int.Parse(node.Attribute("minElement")?.Value ?? "0");
                    int maxElement =
int.Parse(node.Attribute("maxElement")?.Value ?? "0");
                    int repeat = int.Parse(node.Attribute("repeat")?.Value ?? "1");
                    int[] result;
                    if (name.Contains("Arithmetic"))
                    {
                        Console.WriteLine("Arithmetic progression:");
                        int diff = int.Parse(node.Attribute("diff")?.Value ?? "0");
                        for( var length = startLength; length <= maxLength; length+=
diff)
                        {
                            for (int i = 1; i<repeat; i++){
                            result = generator.GetOperationsCount(length,
minElement, maxElement);
                            Console.WriteLine($"{length,6}, {result[1],8}");
                        }
                        }
                        Console.WriteLine("\n");
```

```
                    }
                    else
                    {
                        Console.WriteLine("Geometrical progression:");
                        int znamen = int.Parse(node.Attribute("Znamen")?.Value ??
"2");
                        for( var length = startLength; length <= maxLength; length*=
znamen)
                        {
                            for (int i = 1; i<repeat; i++){
                            result = generator.GetOperationsCount(length,
minElement, maxElement);
                            Console.WriteLine($"{length,6}, {result[1],8}");
                        }
                        }
                    }
                }
            }
            catch (Exception ex)
            {
                Console.WriteLine($"An error occurred: {ex.Message}");
            }
        }
        static void Main()
        {
            if (testing() == false){
                return;
            }
                    Console.Write("Выполнить эксперимент перед началом
программы (yes/no)? ");
```

```
            var answer = Console.ReadLine();
            if (answer == "yes")
                Experiments();

            while (true){
            Console.Write("Введите строки для сортировки, разделяя их пробелом или 'Enter' для завершения: ");

            var line = Console.ReadLine();
            if (string.IsNullOrWhiteSpace(line))
                return;

            var input = line.Split();
            if (input.SequenceEqual(new string[] {""})){
                return;
            }
            var array = new int[input.Length];
            for (int i = 0; i < input.Length; i++){
                array[i] = Int32.Parse(input[i]);
            }
            BubbleSort.Sort(array);
            Console.Write("Отсортированный массив: ");
            foreach (var item in array) Console.Write(item + " ");
            Console.Write("\n\n");
            }
        }
    }


    static bool testing(){
```

```
//Test 1

bool is_passed = true;
var array = new int[] { 5, 4, 3, 2, 1 };
var true_array = new int[] {1, 2, 3, 4, 5};
BubbleSort.Sort(array);

if (!array.SequenceEqual(true_array)){
    Console.WriteLine($"Test 1 isn't passed.");
    Console.WriteLine("Actual:");
    foreach (var item in array) Console.Write(item + " ");
    Console.Write("\n\n");

    Console.WriteLine("Expected:");
    foreach (var item in true_array) Console.Write(item + " ");
    Console.Write("\n\n");

    is_passed = false;
}

//Test 2
array = new int[]{1, 2, 3, 4, 5};
true_array = new int[] {1, 2, 3, 4, 5};
BubbleSort.Sort(array);

if (!array.SequenceEqual(true_array)){
    Console.WriteLine($"Test 2 isn't passed.");
    Console.WriteLine("Actual:");
    foreach (var item in array) Console.Write(item + " ");
```

```
    Console.Write("\n\n");

    Console.WriteLine("Expected:");
    foreach (var item in true_array) Console.Write(item + " ");
    Console.Write("\n\n");

    is_passed = false;
}

//Test 3
array = new int[]{};
true_array = new int[] {};
BubbleSort.Sort(array);

if (!array.SequenceEqual(true_array)){
    Console.WriteLine($"Test 3 isn't passed.");
    Console.WriteLine("Actual:");
    foreach (var item in array) Console.Write(item + " ");
    Console.Write("\n\n");

    Console.WriteLine("Expected:");
    foreach (var item in true_array) Console.Write(item + " ");
    Console.Write("\n\n");

    is_passed = false;
}

//Test 4

array = new int[]{99999999,9999999,1,0,-100};
```

```
            true_array = new int[] {-100, 0, 1, 9999999, 99999999};
            BubbleSort.Sort(array);

            if (!array.SequenceEqual(true_array)){
                Console.WriteLine($"Test 3 isn't passed.");
                Console.WriteLine("Actual:");
                foreach (var item in array) Console.Write(item + " ");
                Console.Write("\n\n");

                Console.WriteLine("Expected:");
                foreach (var item in true_array) Console.Write(item + " ");
                Console.Write("\n\n");

                is_passed = false;
            }

            //Test 5

            array = new int[]{99999999,9999999, 1,0,-100};
            int[] sorted_count = BubbleSort.Sort(array);

            if (sorted_count[0] != 10 || sorted_count[1] != 10){
                Console.WriteLine($"Test 3 isn't passed.");
                Console.WriteLine($"Actual: {sorted_count[0]},
{sorted_count[1]}");

                Console.WriteLine("Expected: 10");
                is_passed = false;
            }
            // Test 6: Все элементы одинаковые
```

```
array = new int[]{7, 7, 7, 7, 7};
true_array = new int[]{7, 7, 7, 7, 7};
BubbleSort.Sort(array);

if (!array.SequenceEqual(true_array)){
    Console.WriteLine("Test 6 isn't passed.");
    is_passed = false;
}

// Test 7: Один элемент
array = new int[]{42};
true_array = new int[]{42};
BubbleSort.Sort(array);

if (!array.SequenceEqual(true_array)){
    Console.WriteLine("Test 7 isn't passed.");
    is_passed = false;
}

// Test 8: Два элемента, не отсортированы
array = new int[]{2, 1};
true_array = new int[]{1, 2};
BubbleSort.Sort(array);

if (!array.SequenceEqual(true_array)){
    Console.WriteLine("Test 8 isn't passed.");
    is_passed = false;
}

// Test 9: Отрицательные числа
```

```
            array = new int[]{-3, -1, -7, -5};
            true_array = new int[]{-7, -5, -3, -1};
            BubbleSort.Sort(array);

            if (!array.SequenceEqual(true_array)){
                Console.WriteLine("Test 9 isn't passed.");
                is_passed = false;
            }

            // Test 10: Большой массив на корректность длины и сортировки
            array = Enumerable.Range(1, 1000).Reverse().ToArray();
            true_array = Enumerable.Range(1, 1000).ToArray();
            BubbleSort.Sort(array);

            if (!array.SequenceEqual(true_array)){
                Console.WriteLine("Test 10 isn't passed.");
                is_passed = false;
            }

            // Test 11: Проверка GetOperationsCount на возврат 2 элементов
            var generator = new ArrayGenerator();
            int[] operations = generator.GetOperationsCount(10, 1, 100);
            if (operations.Length != 2)
            {
                Console.WriteLine("Test 11 isn't passed (GetOperationsCount should return array of length 2).");
                is_passed = false;
            }

            // Test 12: Проверка, что количество сравнений и обменов >= 0
```

```
            if (operations[0] < 0 || operations[1] < 0)
            {
                Console.WriteLine("Test 12 isn't passed (negative operation counts).");
                is_passed = false;
            }
                return is_passed;

            }

        }
}
```